Prolog
outil d'expression et d'exploitation des connaissances.

Pratique avec Prolog II+

PLAN

- Prolog sur un exemple.
- Fondements théoriques et raisonnement de Prolog.
- Récursivité et listes
- Contrôle de la résolution (coupe choix, échec, ...).

C. HANACHI

PROgramming in LOGic

• A. Colmerauer, P. Roussel (1973), Marseille.

• Fondements théoriques du langage : logique du premier ordre, démonstration automatique (Kowalsky, Robinson).

• Prolog peut être considéré comme :

- un démonstrateur de théorèmes ;

- un générateur de SE ;

- un langage de programmation déclaratif ;

- un langage de requêtes.

PROgramming in LOGic

Programmer en Prolog consiste à :

1. déclarer des faits sur des objets

2. énoncer des règles de raisonnement se basant sur ces objets,

3. poser des questions sur ces objets et ces règles.

Prolog sur un exemple
(Syntaxe Edinburgh)

```
/*les assertions ou faits établis*/
        activites(boby_oil, petrole).
        activites(boby_oil, alimentation).
        activites(yakachauffer, alimentation).
        activites(vroom, automobile).
        activites(vroom, alimentation).
        activites(glouton, alimentation).
        activites(glouton, automobile).
        activites(petrole_jr, petrole).

        participations(boby_oil,yakachauffer,60).
        participations(vroom,yakachauffer,40).
        participations(glouton,vroom,40).


/* les règles */

controle_direct(X1,X2) :-   participations(X1,X2,Z), Z>50.

même_secteur(X1,X2) :-    activites(X1, Y1),
                                activites(X2, Y1),
                                Dif(X1,X2).

concurrente(X1,X2) :-           même_secteur(X1,X2),
                                not(controle_direct(X1,X2)),
                                not(contrôle_direct(X2,X1)).

actionnaire(X1):-               participations(X1,_,_).
```

Prolog sur un exemple

QUESTIONS que l'on peut poser à Prolog

La société boby_oil a-t-elle des activités dans le secteur petrolier ?

Question en Prolog : ?- **activites(boby_oil, petrole).**

Réponse Prolog : **{}**

La société yakachauffer a-t-elle des actions?

Question en Prolog : **actionnaire(yakachauffer).**

Réponse Prolog :

Liste des objets X et Y vérifiant le prédicat activite(X,Y)?

Question en prolog : **activites(X, Y).**

Réponse Prolog :

{X=boby-oil, Y=petrole}
{X=boby-oil, Y=alimentation}
{X=yakachauffer, Y=alimentation}
{X=vroom, Y=automobile}
{X=vroom, Y=alimentation}
{X=glouton, Y=alimentation}
{X=glouton, Y=automobile}
{X=petrole-jr, Y=petrole}

Remarque : Prolog donne toutes les solutions possibles au problème : on dit que Prolog est ***non-deterministe.***

Prolog sur un exemple

QUESTIONS que l'on peut poser à Prolog

Quelle est la société qui contrôle yakachauffer ?

Question en Prolog : **controle_direct(X,yakachauffer).**

Réponse Prolog : **{X=boby_oil}**

Quelles sont les sociétés qui ont une participation supérieure à 30 dans la société yakachauffer ?

Question en prolog :

participation(X,yakachauffer, P), P> 30.

Réponse prolog :

{X=boby_oil, P=60}
{X=vroom, P=40}

*C*ouples d'entreprises concurrentes ?

Question en prolog : **concurrente(X,Y).**

Réponse en prolog ? : .....

Fondements théoriques et raisonnement

1. PASSAGE D'UNE FORMULE LOGIQUE À UN PROGRAMME PROLOG

Éléments du langage des prédicats

• Termes : constantes (a, b, c), variables (X, Y, Z, Xa, ...), Fonction(f,g,h,...)

• Prédicat : symbole(P, Q, participations, ...) qui prend un ou plusieurs arguments et retourne l'une des valeurs vrai ou faux.

• Quantificateur : $\forall, \exists$

• Les parenthèses : (,)

• Les connecteurs : ET, OU, non, $\Rightarrow$, $\Leftrightarrow$

D'autres notions

Définition : Si t1, t2, ...tn sont des termes et P un prédicat alors P(t1,t2,...,tn) est **une formule atomique**.

Définition : on appelle **littéral** à la fois les formules atomiques(littéral positif) et leur négation (littéral négatif).

Définition : Une **Formule Bien Formée** (FBF) est une expression logique définie à l'aide des règles suivantes :

- une formule atomique est une FBF,
- Si F1 et F2 sont des FBF alors (F1 ET F2), (F1 OU F2), (F1 $\Rightarrow$F2), (F1$\Leftrightarrow$F2), (non F1) sont des FBF
- Si F est une FBF alors ($\forall$ X)(F), ($\exists$ X)(F) sont des FBF.

Fondements théoriques et raisonnement

Notions de validité et d'inconsistance

Définition : Une formule est dite valide si elle prend la valeur vrai pour toutes les interprétations possibles.

Définition : Une formule est inconsistante si et seulement si elle prend la valeur faux dans toute interprétation.

Définiton : Une formule Q résulte logiquement d'un ensemble de formules P si toute interprétation qui satisfait P satisfait également Q. Cette propriété équivaut à l'inconsistance de l'ensemble P $\cup$ {non Q}.

Propriétés

• Identique à celle du calcul des propositions (lois de Morgan, distributivité, commutativité, associativité, + divers règles transformations)
• non ((∀ X) P(X)) = (∃ X) (non P(X))
• non ((∃ X)P(X)) = (∀ X) (non P(X))
• (∀ X) (G(X) ET H(X)) = ((∀ X) G(X) ET (∀ Y) H(Y))
• (∃ X) (G(X) OU H(X)) = ((∃ X) G(X) OU (∃ Y) H(Y))

Fondements théoriques et raisonnement

Passage de FBF à des clauses de horn (cf. cours de logique)
• mise sous forme normale prenexe :élimination des connecteurs, accoler les Not aux atomes concernés, déplacements des quantificateurs à gauche.

• passage sous forme de clauses : éliminer les quantificateurs existentiels(skolemisation), éliminer les quantificateurs universels(qui restent implicites), passage sous forme normale conjonctive (conjonction de clause).

Définition : On appelle **clause** (en logique des prédicats) toute disjonction de littéraux. Soit une expression de la forme :
A1 .....OU An OU (non B1) OU .....(non Bm), où les Ai et les Bi sont des prédicats.

Définition : On appelle **clause de Horn** une clause dont un seul élément au plus est un littéral positif. Soit une expression de la forme : A OU (non B1) OU ... (non Bm).

Remarque:
Une telle clause est équivalente à l'implication suivante :

B1 ET B2 ET ...Bm $\Rightarrow$ A

Ce qui correspond à des règles de la forme :

Si B1 ET B2 ET ....Bm Alors A

ou en Prolog II+: **A :-B1,B2, ...,Bn .**

Fondements théoriques et raisonnement

Règles de Déduction

- Modus Ponens : ((P $\Rightarrow$ Q) ET P) $\Rightarrow$ P
- Modus Tollens : ((P $\Rightarrow$ Q) ET (non Q)) $\Rightarrow$ (non P)
- Spécialisation universelle : ((∀ X) P(X)) $\Rightarrow$ P(A)

- Le principe de Résolution de Robinson (cf. cours de logique) :
  Soient :
  P={C1,C2,...,Cn} : ensembles des clauses constituant les faits et les règles d'un programme Prolog.
  Q : le but Prolog (conjonctions de littéraux positifs).

  Le principe va consister à essayer de déduire la clause vide à partir de {C1, C2,...,Cn, non Q}. Ce qui permet de réfuter (non Q) et de prouver Q.

Fondements théoriques et raisonnement

Exemple :

Soit le programme Prolog suivant :

1 contrôle(S2,S1).
2 contrôle(S3,S2).
3 contrôle_intermédiare(X,Y):-contrôle(X,Z),contrôle(Z,Y).

Soit la question suivante :

4 contrôle_intermédiare (X,Y).

L'instruction 3 représente la formule des prédicats :

$(\forall X), (\forall Y), (\forall Z)$ Non contrôle(X,Z) ou Non contrôle(Z,Y) ou contrôle_intermédiare(X,Y)

qui équivaut à

$(\forall X), (\forall Y), (\forall Z)$ (contrôle(X,Z) et contrôle(Z,Y)) $\Rightarrow$ contrôle_intermédiare(X,Y)

L'instruction 4 représente la formule des prédicats :

$(\forall X), (\forall Y)$ Non contrôle_intermédiare(X,Y)

c'est à dire la négation de

$(\exists X), (\exists Y)$ contrôle_intermédiare(X,Y)

Si par résolutions successives, on dérive la clause vide, on aura réfuté *$(\forall X), (\forall Y)$ Non contrôle_intermédiare(X,Y)* et donc prouvé que *$(\exists X), (\exists Y)$ contrôle_intermédiare(X,Y).* Les instanciations opérées sur X et Y pour conclure cette réfutation sont une réponse à la question:*$(\exists ?X), (\exists ?Y)$ contrôle_intermédiare(X,Y).*

Fondements théoriques et raisonnement

• un programme Prolog est un ensemble de **clauses de horn** ayant trois formes possibles :

B:-A1,A2,...,An . Règle.

B(:-). Faits.

(:-)A1,A2,...,An Questions (But).

• Plusieurs définitions d'un même prédicat (Expression du "ou") :

grand_parent (X,Y):-grand_mere(X,Y) .

grand_parent (X,Y):-grand_pere(X,Y).

• Plusieurs littéraux peuvent appartenir à une même queue de clause (expression du "et") :

grand_pere(X,Y):-pere(X,Z),parent (Z,Y).

• les variables sont locales à la clause.

Fondements théoriques et raisonnement

Pour résoudre un problème, l'interprète Prolog développe un algorithme de résolution particulier **LUSH résolution** (résolution linéaire) :

• parcours d'une disjonction (**ou**) de clauses de **haut en bas** ;

• parcours d'une conjonction (et) de clauses de **gauche à droite** ;

• retour arrière systématique (backtracking) pour essayer d'autres règles ou assertions CHRONOLOGIQUEMENT (retour au dernier point de choix dans le graphe de recherche).

• Vision système expert : chaînage arrière, avec régime par tentative, choix de la première règle déclenchable d'abord(dans l'ordre de l'écriture), traitement du problème le plus récent d'abord(stratégie en profondeur).

Fondements théoriques et raisonnement

Sens logique des clauses

la règle :

B:-A1,A2,...,An .

s'interprète logiquement B est vérifié si A1A2...An est vérifié ∀ X1, X2, Xk variables apparaissant dans B, A1,..., An.

L'assertion :

B(:-).

s'interprète logiquement B est toujours vérifié ∀ (X1,..., Xk) appartenant à B.

Sens procédural des clauses (ce que fait l'interprète)

La règle B:-A1,A2,...,An . s'interprète procéduralement pour résoudre B, il faut résoudre successivement A1, ..., An.

L'assertion

B(:-).

s'interprète comme le problème B est résolu.

2. L'unification

Qu'est-ce ?

• correspond à l'étape de filtrage pour les moteurs basés sur la logique des prédicats.
• permet de déterminer si une règle est déclenchable : si sa partie déclencheur est compatible avec la base de faits.

Définition : Deux termes T et U sont unifiables ssi il existe une ou plusieurs substitutions qui, appliqués à la fois et dans le même ordre à T et U rendent ces deux termes identiques.

Substitutions autorisées :

1. une variable peut être remplacée par une constante.
2. une variable peut être remplacée par une variable.
3. une variable peut être remplacée par une expression fonctionnelle si celle-ci ne contient pas cette variable.

Fondements théoriques et raisonnement

Exemple :

P(X, f(X,a),Y)et P(X,Y,g(a,b))

Algorithme :

1. parcours de l'arbre jusqu'à obtention d'une paire de discordance ici :

f(X,a) et Y

2. détermination d'une substitution pour éliminer cette discordance, ici

remplacer Y par f(X,a), on note {f(X,a) / Y}

3. application de la substitution, ici on obtient :

P(X,f(X,A),f(X,A)) et P(X,f(X,a),g(a,b))

4. on recommence les étapes 1), 2), 3) jusqu'à que les deux expressions soient les même ou impossibilité d'unifier !

Exemples:

unifier (X=a) et (a=Y)

unifier P(X,a) et P(f(Y),Y)

Fondements théoriques et raisonnement

(graphe de recherche sur un exemple).

/*les assertions ou faits établis*/

activites(boby_oil, petrole).
activites(boby_oil, alimentation).
activites(yakachauffer, alimentation).
activites(vroom, automobile).
activites(vroom, alimentation).
activites(glouton, alimentation).
activites(glouton, automobile).
activites(petrole_jr, petrole).

participations(boby-oil,yakachauffer,60).
participations(vroom,yakachauffer,40).
participations(glouton,vroom,40).

/* les règles */

controle_direct(X1,X2) :- participations(X1,X2,Z), Z>50.

Question :
controle_direct(S,yakachauffer), activite(S,petrole).

Récursivité et Listes

• Recherche de chemin dans un graphe

```
arc(a,b).
arc(b,c).
arc(c,e).
arc(c,d).
arc(a,e).
```

Chemin entre X et Y ? : chemin(X,Y).
Deux cas :

```
chemin(X,Y):-   arc(X,Y).
chemin(X,Y):-   arc(X,Z),
                chemin(Z,Y).
```

• Longueur du chemin :

```
chemin(X,Y, L):-  arc(X,Y), L is 1.
chemin(X,Y,L):-   arc(X,Z)
                      chemin(Z,Y,L1),
                      L is L1 +1.
```

Exercice : Factorielle, Fibonacci, ...

Récursivité et Listes

• [T|Q] : T est la tête de la liste, Q est la queue.

• la liste vide : []

• [1,2,3,4] = [1| [2,3,4]]=[1| [2| [3,4]] = ...
= [1| [2| [3| [4| []]]]]

• l'ordre est important.

• [E1| L1] =[E2 | L2] ssi E1 = E2 et L1 = L2

• Exemple d'unification :

		résultats
[X | [2,3,4]]	[1| L]	X=1 et L=[2,3,4]
[X | L]	LL	LL=[X | L]
[X]	[1|L]	X=1 et L=[]
[1| L]	[2 | M]	échec
[]	[X]	échec

• La liste [] n'est unifiable qu'avec elle même.

```
appartient(E,[E|_]).
appartient(E,[_|L]):-appartient(E,L).

longueur([ ],0).
longueur([_|L],N):-longueur(L,NN), N is NN+1.
```

Exercices :
• prédicat qui calule le Max d'une liste.
• prédicat qui retourne vrai si Les listes X et Y sont disjointes.

CONTROLE DE LA RESOLUTION
(déterminisme versus non-déterminisme)
(non-lisibilité versus lisibilité)

• Gestion du retour arrière

Définition : On appelle coupe-choix ou CUT et on note "!" un prédicat qui provoque la suppression des choix possibles restants pour les littéraux allant de la tête de la clause courante incluse jusqu'à ce prédicat.

Utilisation :

1) sert à éviter des explorations inutiles ;
2) sert à contrôler le déroulement d'un programme.

Exemple 1 : éviter des calculs inutiles !

définition du maximum :

```
max1(X1,X2,M):-X1<=X2,M=X2.
max1(X1,X2,M):-X1>X2,M=X1.
```

ou

```
max2(X1,X2,M):-X1<=X2,!, M=X2.
max2(X1,X2,M):-X1>X2,M=X1.
```

ou

```
max3(X1,X2,M):-X1<=X2,!, M=X2.
max3(X1,X2,M):-M=X1.
```

ou

```
max4(X1,X2,X2):-X1<=X2,!.
max4(X1,_,X1).
```

Exemple 2 : coupe choix pour définir l'alternative

```
if_then_else(P,Q,R) :-P, !, Q.
if_then_else(P,Q,R) :-R.
```

CONTROLE DE LA RESOLUTION
(déterminisme versus non-déterminisme)
(non-lisibilité versus lisibilité)

• Gestion de l'échec

la primitive **fail** provoque l'échec d'une clause.

a:-b,c,fail.

si b et c sont évalués à vrai, l'exécution de fail va rendre le résultat final FAUX pour l'exécution de la clause.

Exemple : expression du NOT.

```
not_cond(X) : -cond(X),!,fail.
not_cond(X).
```

Explication : ou bien cond(X) est satisfait, et alors le cut s'efface et coupe le choix en suspens correspondant à la deuxième règle (Elle ne sera donc pas envisagée). Le fail donne alors un échec. Ou bien cond(x) n'est pas satisfait, alors la deuxième règle est essayé : ce qui donne un succés. not_cond a bien la valeur opposée de cond.

Exemple : expression de la différence

```
diff(X,X):-!,fail.
diff(X,Y).
```